IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In Re Application of:

Gouriou, et al.

Group Art Unit: 2193

Serial No.: 10/608,330

Examiner: Francis, Mark

Filed: June 27, 2003

Docket No. 200206152-1

For: **Method and Apparatus for Controlling Execution of a Child Process Generated by a Modified Parent Process**

APPEAL BRIEF UNDER 37 C.F.R. § 41.37

Mail Stop: Appeal Brief-Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450

Sir:

This Appeal Brief under 37 C.F.R. § 41.37 is submitted in support of the Notice of Appeal filed June 6, 2007, responding to the Final Office Action mailed December 6, 2006.

It is not believed that extensions of time or fees are required to consider this Appeal Brief. However, in the event that additional extensions of time are necessary to allow consideration of this paper, such extensions are hereby petitioned under 37 C.F.R. § 1.136(a), and any fees required therefor are hereby authorized to be charged to Deposit Account No. 08-2025.

I. Real Party in Interest

The real party in interest is Hewlett-Packard Development Company, LP, a limited partnership established under the laws of the State of Texas and having a principal place of business at 20555 S.H. 249 Houston, TX 77070, U.S.A. (hereinafter "HPDC"). HPDC is a Texas limited partnership and is a wholly-owned affiliate of Hewlett-Packard Company, a Delaware Corporation, headquartered in Palo Alto, CA. The general or managing partner of HPDC is HPQ Holdings, LLC.

II. Related Appeals and Interferences

There are no known related appeals or interferences that will affect or be affected by a decision in this Appeal.

III. Status of Claims

Claims 1-9, 12, 14-28, 31, 33, 34, 36 have been canceled leaving claims 10, 11, 13, 29, 30, 32, 35, and 37-39 remaining. Although each of those claims is indicated as being rejected under 35 U.S.C. § 102(e) in the final Office Action, the final Office Action also indicated that claim 29 is allowed. In a telephone conference with the Examiner, Applicant confirmed that claim 29, and claims 30 and 32 that depend from claim 29, are in fact allowed. Therefore, only the final rejections of claims 10, 11, 13, 35, and 37-39 are appealed and addressed in this Appeal Brief.

IV. Status of Amendments

This application was originally filed on June 27, 2003, with twenty-eight (28) claims. In a Preliminary Amendment filed September 16, 2003, Applicant amended claims 1, 3, 5-8, 10, 11, 13, 16, 17, canceled claims 9, 20-28, and added new claims 29-36. In a Response filed September 14, 2006, Applicant amended claims 10, 11, 13, 29, 35, canceled claims 1-9, 12, 14-28, 31, 33, 34, 36, and added new claims 37-39. In a Response filed February 5, 2007, Applicant amended claim 37.

All of the above-identified amendments have been entered and no other amendments have been made to any of claims 10, 11, 13, 29, 30, 32, 35, and 37-39. The claims in the attached Claims Appendix (see below) reflect the present state of those claims, including allowed claims 29, 30, and 32.

V. Summary of Rejected Claims

The claimed inventions are summarized below with reference numerals and references to the written description ("specification") and drawings. The subject matter described in the following appears in the original disclosure at least where indicated, and may further appear in other places within the original disclosure.

Independent claim 10 describes a method for controlling the execution of a child process created from a parent process. The method comprises instrumenting a parent process. *Applicant's specification*, paragraphs 0043, 0057, and 0058; Figure 4, items 402-408. The method of claim 10 further comprises monitoring execution of the parent process with a process monitor to collect information as to the run-time behavior of the parent process. *Applicant's specification*, paragraphs 0052 and 0059; Figure 3, item

320; Figure 4, item 410. The method of claim 10 further comprises, before a vfork system call is executed, receiving with the process monitor indicia from the parent process that a vfork system call will be executed by the parent process. *Applicant's specification*, paragraphs 0059, 0064, and 0070; Figure 4, item 412; Figure 5, item 506; Figure 7, item 702. The method of claim 10 further comprises suspending execution of the parent process. *Applicant's specification*, paragraph 0064; Figure 5, item 510. The method of claim 10 further comprises extracting with the process monitor a process identifier from the indicia, the process identifier identifying a child process to be generated by the parent process when the parent process executes the vfork system call. *Applicant's specification*, paragraphs 0059 and 0070; Figure 4, item 412; Figure 7, item 706. The method of claim 10 further comprises setting with the process monitor a process monitor thread to observe trace events generated by the child process. *Applicant's specification*, paragraphs 0059 and 0070; Figure 4, item 416; Figure 7, item 708. The method of claim 10 further comprises resuming execution of the parent process to enable the parent process to execute the vfork system call. *Applicant's specification*, paragraphs 0065 and 0070; Figure 5, item 520; Figure 7, item 712. The method of claim 10 further comprises again suspending execution of the parent process. *Applicant's specification*, paragraphs 0064 and 0070; Figure 5, item 514; Figure 7, item 704.

Independent claim 35 describes a computer readable memory (220, Figure 2) that stores a system (300, Figures 2 and 3) that comprises a parent process (350, Figure 3) configured to, before a vfork call is executed by the parent process, generate a pre-fork event that contains a process identifier of a child process that will be spawned from the parent process when the vfork system call is executed by the parent process. *Applicant's specification*, paragraphs 0059, 0064, and 0070; Figure 4, item 412; Figure 5, item 506; Figure 7, item 702. In addition, the system of claim 35 further comprises a process monitor (320, Figure 2) configured to receive the pre-fork event and process identifier before the vfork system call is executed by the parent process, suspend execution of the parent process, and generate a process monitor thread that enables observation of trace events generated by the child process. *Applicant's specification*, paragraphs 0059, 0064, and 0070; Figure 4, items 412 and 416; Figure 5, item 510; Figure 7, items 706 and 708.

VI. Grounds of Rejection to be Reviewed on Appeal

The following grounds of rejection is to be reviewed on appeal:

1. Claim 35 has been rejected under 35 U.S.C. § 101 because the claimed invention is directed to non-statutory subject matter.

2. Claims 10, 11, 13, 35, and 37-39 have been rejected under 35 U.S.C. § 102(e) as being anticipated by *Bennett* (U.S. Pat. No. 7,114,104).

VII. Arguments

The Appellant respectfully submits that Applicant's claims are directed to statutory subject matter under 35 U.S.C. § 101 and are not anticipated under 35 U.S.C. § 102. Applicant therefore respectfully requests that the Board of Patent Appeals ("the Board") overturn the rejections of Applicant's claims at least for the reasons discussed below.

A. Claim Rejections - 35 U.S.C. § 101

Claim 35 has been rejected under 35 U.S.C. § 101 as being drawn to non-statutory subject matter. In particular, the Examiner stated in the final Office Action that claim 35 does not "require use of hardware or a tangible medium." Applicant traverses.

Applicant disagrees that claim 35 does not "require use of hardware or a tangible medium" given that the invention claimed in claim 35 *is* such a medium. In particular, claim 35 is explicitly directed to a "computer readable memory". Such a "memory" *is* a hardware component and clearly qualifies as a "manufacture" under 35 U.S.C. § 101. *See* 35 U.S.C. § 101. Accordingly, Applicant respectfully submits that claim 35 is drawn to statutory subject matter.

Regarding the Examiner's position that a "memory" cannot store a "system", Applicant notes that the term "system" does not necessarily require hardware. For instance, a suite of software could rightly be described as a "system". Regardless, if the Examiner believed that the term "system" was indefinite or ambiguous, the Examiner should have addressed the issue under 35 U.S.C. 112, second paragraph, not 35 U.S.C. § 101. Applicant's use of the term "system" does not change the fact that claim 35 is explicitly drawn to a "computer readable memory" and therefore has no bearing on whether claim 35 is or is not directed to statutory subject matter.

In the Advisory Action, the Examiner maintains that claim 35 is directed to non-statutory subject matter irrespective of the fact that it is explicitly drawn to a "computer readable medium" because the body of the claim recites "software means". In response, Applicant believes that the Examiner is mistaken as to what is required under the law. Specifically, 35 U.S.C. § 101 requires that a claim be directed to one or more of four different statutory categories of inventions, including any useful "process, machine, manufacture, or composition of matter." If the claimed invention is directed to one of those categories and further does not merely comprise an abstract idea, a law of nature, or a natural phenomenon, the claim is directed to statutory subject matter. *See USPTO Official Gazette Notice* of November 22, 2005. In the present case, the claimed "computer readable memory" is a "manufacture." Furthermore, the claimed "computer readable memory" is a physical thing and cannot reasonably be considered to comprise "an abstract idea, a law of nature, or a natural phenomenon." Therefore, claim 35 is directed to patentable subject matter regardless of whether the claim includes recitations of "software means".

As further evidence that claim 35 is directed to statutory subject matter, suppose Applicant had explicitly directed claim 35 to a specific type of "computer readable medium", such as a "desktop computer." Clearly a claim to a "desktop computer" would be statutory, irrespective of whether it mentioned "software means".

In view of the above, Applicant respectfully submits that claim 35 is directed to statutory subject matter and respectfully requests that the Board overturn the Examiner's rejection.

B. Claim Rejections - 35 U.S.C. § 102(e)

Claims 10, 11, 13, 35, and 37-39 have been rejected under 35 U.S.C. § 102(e) as being anticipated by *Bennett* (U.S. Pat. No. 7,114,104). Applicant respectfully traverses this rejection.

It is axiomatic that "[a]nticipation requires the disclosure in a single prior art reference of each element of the claim under consideration." *W. L. Gore & Associates, Inc. v. Garlock, Inc.*, 721 F.2d 1540, 1554, 220 U.S.P.Q. 303, 313 (Fed. Cir. 1983). Therefore, every claimed feature of the claimed invention must be represented in the applied reference to constitute a proper rejection under 35 U.S.C. § 102(e).

In the present case, not every feature of the claimed invention is represented in the Bennett reference. Applicant discusses the Bennett reference and Applicant's claims in the following.

1. The Bennett Disclosure

Bennett discloses a process fault monitor for Unix systems. *Bennett*, column 2, lines 33-53. The fault monitor watches operating system resources associated with the processes to determine when an interruption occurs, evaluates the status of the process, and selectively signals the interruption, for example, to an event router that may in turn signal the interruption to a fault handling resource. *Id.* In some embodiments, the fault monitor uses a fault detection driver to identify the start of new processes and locate the operating system resources associated with those processes for monitoring. *Id.*

2. Applicant's Claims

Bennett fails to teach several of Applicant's explicit claim limitations. Applicant discusses some of those claim limitations in the following.

a. Claims 10, 11, 13, and 37

Applicant's independent claim 10 provides as follows (emphasis added):

> 10. A method for controlling the execution of a child process created from a parent process, the method comprising:
>
> instrumenting a parent process;
>
> monitoring execution of the parent process with a process monitor to collect information as to the run-time behavior of the parent process;
>
> before a vfork system call is executed, receiving with the process monitor indicia from the parent process that a vfork system call will be executed by the parent process;
>
> suspending execution of the parent process;
>
> extracting with the process monitor a process identifier from the indicia, the process identifier identifying a child process to be generated by the parent process when the parent process executes the vfork system call;
>
> setting with the process monitor a process monitor thread to observe trace events generated by the child process;

resuming execution of the parent process to enable the parent process to execute the vfork system call; and

again suspending execution of the parent process.

(i) Failure to Teach "Instrumenting" a Parent Process

As a first matter, Bennett does not teach "instrumenting" a parent process. Although Bennett generally describes an "OS fork routine 224" that creates a child process from a parent process in column 6, lines 30-50, nowhere is instrumenting of the parent process, or the child process for that matter, discussed.

In the Advisory Action, the Examiner reiterates his argument that column 6, lines 30-42 of the Bennett reference teach "instrumenting a parent process". That portion of Bennett's disclosure provides as follows:

> The OS fork routine 224 is a system routine invoked by the OS to start a new process. The new process, or child process, is an exact copy of the calling process, or parent process. The child process inherits most of its attributes from the parent process, but it receives its own process ID, entries within the process control data structure, associated memory allocations, and a few other unique attributes. The OS fork routine 224 returns the process ID of the child process to the parent process. The OS fork routine 224 is used by the OS to start almost all new processes. Some OS include multiple fork routines with associated system calls. For example, Solaris includes fork( ), vfork( ), and fork1( ) system calls for accessing three variations on the OS fork routine 224.

Bennett, column 6, lines 30-42. As can be appreciated from the above excerpt, Bennett describes a routine 224 that starts a new process (i.e., a child process) from a parent process. Despite that teaching, Bennett fails to describe "instrumenting" the parent process. Instead, Bennett merely states that a child process is created as an exact copy of the parent process. No modification of the parent process is described.

In the Advisory Action, the Examiner argues that Bennett teaches in column 6, lines 30-42: (i) "instrumenting" a child process by creating it from a parent process and (ii) that such creation of the child process somehow inherently constitutes instrumenting the parent process. In response, Applicant first notes that merely creating a child process that is an exact copy of a parent process is not "instrumenting" a process within the well-established meaning of the term in the art. As is well understood by persons having ordinary skill in the art, "instrumenting" is a technique in which *code is inserted into* an process, for example to collect information regarding the process' execution. Applicant describes such instrumenting in paragraph 0050 of Applicant's specification. There, Applicant states that an "instrumentation engine 310" of a "software monitor 300" "inserts code into the target process". *Applicant's specification*, paragraph 0050. Clearly, Bennett teaches nothing of the sort in column 6, lines 30-42 or elsewhere in his disclosure.

Furthermore, even if Bennett did in fact teach instrumenting the child process, there is no reason to presume that the parent process from which the child process was created would likewise be instrumented. Specifically, if the child process were spawned from the parent process and then instrumented to collect information about the child process' execution, the parent process would not likewise become instrumented. In other words, inserting code into a child process would not automatically result in insertion of code into its parent process.

In view of the above, it is clear that Bennett does not actually teach "instrumenting a parent process" as is explicitly required by Applicant's claim 10.

(ii) Failure to Teach "Before a vfork System Call is Executed, Receiving . . . Indicia from the Parent Process that a vfork System Call will be Executed by the Parent Process"

Second, Bennett does not teach "before a vfork system call is executed, receiving with the process monitor indicia from the parent process that a vfork system call will be executed by the parent process". Although Bennett generally discusses child processes and mentions the term "vfork( )" in column 6, lines 30-50, which were relied upon by the Examiner in the final Office Action, that portion of Bennett's disclosure says nothing about a parent process indicating that a vfork system call will be executed by the parent process.

In the Advisory Action, the Examiner argues that Bennett teaches in column 6, lines 30-42 that: (i) a fault detection driver will accumulate process Ids for each new process and (ii) "Solaris" includes vfork( ) system calls. Applicant again reproduces Bennett's column 6, lines 30-42 below for purposes of discussion:

> The OS fork routine 224 is a system routine invoked by the OS to start a new process. The new process, or child process, is an exact copy of the calling process, or parent process. The child process inherits most of its attributes from the parent process, but it receives its own process ID, entries within the process control data structure, associated memory allocations, and a few other unique attributes. The OS fork routine 224 returns the process ID of the child process to the parent process. The OS fork routine 224 is used by the OS to start almost all new processes. Some OS include multiple fork routines with associated system calls. For example, Solaris

includes fork( ), vfork( ), and fork1( ) system calls for accessing three variations on the OS fork routine 224.

Bennett, column 6, lines 30-42. As a first matter regarding the Examiner's argument, Bennett says nothing about a "fault detection driver" in column 6, lines 30-42. Therefore, Applicant cannot follow the Examiner's logic as to that part of his argument. Second, although Bennett does state that "Solaris" includes vfork( ) system calls, that statement neither teaches nor suggests the action of receiving "indicia from the parent process that a vfork system call will be executed by the parent process" "before a vfork system call is executed". Indeed, even if one were to assume that Bennett disclosed a parent process providing an indication when a vfork system call is executed. Bennett is completely silent as to providing an indication that such a system call *is about to happen*, that is *before* it actually does happen.

The Examiner further argues in the Advisory Action that Bennett teaches in column 2, lines 45-53 a fault monitor that signals interrupts. Even assuming that Bennett provides such a teaching, such a interrupt signal would be an indication of an event that *has already occurred*. Therefore, such a signal cannot be considered to be an indication that something that *will* occur.

(iii) Failure to Teach Suspending Execution of a "Parent Process"

Third, Bennett does not teach the action of "suspending execution of the parent process". Although, as described above, Bennett describes monitoring a process to identify when a fault occurs, such monitoring clearly does not constitute the affirmative action of "suspending" execution. In Bennett's system, such a suspension of parent process execution can be monitored if it occurs (e.g., due to a fault) but cannot be affirmatively caused.

In the Advisory Action, the Examiner now argues that Bennett teaches a "monitor suspension control message" in column 10, lines 25-35. In response, Applicant notes that Bennett's control message "suspends the monitoring thread not a parent process". *Bennett*, column 10, lines 29-31. Such a disclosure is not a teaching of suspending execution "of the parent process" as is explicitly required by Applicant's claim 10.

(iv) Failure to Teach "Extracting . . . a Process Identifier . . . Identifying a Child Process to be Generated . . . when the Parent Process Executes the vfork System Call"

Fourth, Bennett does not teach "extracting with the process monitor a process identifier from the indicia, the process identifier identifying a child process to be generated by the parent process when the parent process executes the vfork system call". As an initial matter, the "indicia" recited in the above limitation of claim 10 is the indicia that was, as recited in the claim, provided from the claimed "parent process". As noted above in subsection (b), Bennett fails to teach a parent process providing such an indication. It therefore logically follows that Bennett cannot be said to teach "extracting with the process monitor a process identifier from the indicia, the process identifier identifying a child process to be generated by the parent process when the parent process executes the vfork system call".

Regardless, Applicant notes that, although Bennett describes "process IDs" in column 8, lines 30-50, which were relied upon by the Examiner in the final Office Action, Bennett does not describe the process ID as "identifying a child process" or that the child process is "to be generated", i.e., that does not yet exist.

In the Advisory Action, the Examiner maintains his argument that Bennett discloses "extracting with the process monitor a process identifier from the indicia, the process identifier identifying a child process to be generated by the parent process when the parent process executes the vfork system call" in column 8, lines 33-45. That portion of Bennett's disclosure provides as follows:

> It is then left to the fault monitor 212 to determine for which processes it has already started a monitoring thread and for which processes a new thread should be started. In one embodiment, the fault monitor 210 sends a read call to the fault detection driver 214 as soon as it has returned and handled the prior batch of process IDs. In this way, new process IDs are returned in the next read call or immediately fulfill a blocked read call. In an alternate embodiment, receipt of a new process ID from a new process will automatically generate a new process ID message to the fault monitor 212 from the fault detection driver 214. In another alternate embodiment, the fault monitor 212 will issue a periodic read call to the fault detection driver 214 to determine if any new processes have begun.

Bennett, column 8, lines 33-45. As is abundantly clear from the above excerpt, Bennett says nothing whatsoever about a process identifier that identifies "a *child process to be generated* by the parent process". To the contrary, Bennett describes receiving a "new process ID" from a new process, i.e., a new process that is already in existence. Further to the contrary, Bennett describes determining if any new processes "have begun," i.e., are already in existence.

Also in the Advisory Action, the Examiner argues that Bennett column 6, lines 30-45 teaches "identifying a child process to be generated by the parent process." As can be readily determined through reference to the above-provided reproductions of Bennett's column 6, lines 30-45, Bennett says nothing at all of identifying a child process that is "to be generated", i.e., that has not yet been generated. *See Bennett*, column 30-45.

(v) Failure to Teach "Setting . . . a Process Monitor Thread to Observe Trace Events Generated by the Child Process"

Fifth, Bennett does not teach "setting with the process monitor a process monitor thread to observe trace events generated by the child process". Although Bennett generally describes using threads to monitor processes, nowhere does Bennett state that such threads are processed "to observe trace events generated by [a] child process". Column 9, lines 40-65 of the Bennett reference, which were relied upon by the Examiner in the final Office Action, do not mention observation of "trace events" or "child events".

In the Advisory Action, the Examiner now argues that Bennett teaches "setting with the process monitor a process monitor thread to observe trace events generated by the child process" in column 7, lines 40-64. That portion of Bennett's disclosure provides as follows:

> FIG. 3 describes the monitoring and event generation process for an example process n 250. The fault manager 210 relies on the fault monitor 212 to drive operation of the monitoring process. The fault detection driver

214 and the event router 216 support the operations of the fault monitor 212. The fault monitor 212 interacts with the OS process control file 254 and the OS status file 256 associated with the process n 250 to establish monitoring conditions, identify interruptions, and analyze those interruptions.

The fault monitor 212 initiates fault monitoring by issuing an open call 302 to the fault detection driver 214. When the fault detection driver 214 receives the open call 302, it begins monitoring for new processes. When a process starts 304, the process ID n 252 is generated for controlling and managing the process. The process ID n 252 is communicated 306 to the fault detection driver 214. The process ID n 252 is intercepted and stored by the fault detection driver 214. The fault detection driver 214 may store the process IDs in any number of data structures. For example, the process IDs may be stored in a linked list in a known storage location. The linked list may provide a fixed number (e.g., 10) of storage locations. The fault detection driver 214 using linked lists provides storage beyond the fixed number by including a pointer to a new storage location allocated for another process ID list.

Bennett, column 7, lines 40-64. As can be appreciated from the above excerpt, not only does Bennett fail to teach "setting . . . a process monitor thread to observe trace events generated by the child process", Bennett fails to even mention a process monitor thread. It is not understood how the Examiner can reasonably state that Bennett teaches setting a process monitor thread to observe trace events generated by a child process in column 7, lines 40-64 when Bennett does not even identify a monitor thread in that portion of Bennett's disclosure.

In addition to failing to describe a monitor thread, Bennett further does not mention monitoring “trace events” of a “child process”. Indeed, column 7, lines 40-64 does not even contain either of the terms “trace event” or “child process”. Again, it is not understood how the Examiner can reasonably allege that Bennett teaches setting with the process monitor a process monitor thread to observe trace events generated by the child process in column 7, lines 40-64 when he does not even mention trace events or child processes. Furthermore, although an “interruption” may correctly be referred to as an “event”, an “interruption” is not a “trace event” as is explicitly required by Applicant’s claim 10.

(vi) Failure to Teach “Resuming Execution of the Parent Process to Enable the Parent Process to Execute the vfork System Call”

Sixth, Bennett does not teach “resuming execution of the parent process to enable the parent process to execute the vfork system call”. As described above, column 6, lines 30-60 of the Bennett reference, which were identified by the Examiner in the final Office Action, generally discuss the creation of processes, not the resumption of execution of processes.

In the Advisory Action, the Examiner now argues that Bennett teaches in column 10, lines 55-64 “resuming execution of the parent process to enable the parent process to execute the vfork system call”. That portion of Bennett’s disclosure provides as follows:

> This determination is made based upon logic within the fault monitor 212. In this instance, the fault monitor 212 writes a control message back to the process control file 254 to cause it to continue processing and re-writes the monitor suspending control message. Monitoring continues as it had prior to the interruption. The fault monitor 212 may or may not raise 338 an appropriate event to the event router depending on whether or not the interruption was of a variety the user wishes to track (in spite of the fact that it may be handled without a core dump).

Bennett, column 10, lines 56-65. As can be appreciated from the above, Bennett does not in fact teach "resuming execution of the parent process to enable the parent process to execute the vfork system call" in column 10, lines 56-65. Instead, Bennett merely describes continuing monitoring after an interruption occurs. Not described is the affirmative action of "resuming execution" of a parent process. Moreover, not described is resuming execution "to enable the parent process to execute the vfork system call". Indeed, column 10, lines 56-65 is silent as to any "system call", including a "vfork" system call.

(vii) Failure to Teach "Suspending Execution of the Parent Process"

Seventh, Bennett does not teach "suspending execution of the parent process". Column 9, lines 43-60 of the Bennett reference, which were relied upon by the Examiner in the final Office Action, provide as follows:

> Based upon the monitoring preferences, one or more control messages are written 318 to the process control file 254. Example control messages

> include signal monitoring, exit tracing, process state monitoring, and monitor suspending. In the embodiment shown, a signal monitoring control message is stored 320 in the process control file 254. The signal monitoring control message specifies which Unix signals to monitor for the target process. The list of monitored signals is derived from the monitoring preferences. In general, the signal monitoring control message will identify a plurality of signals that, if not handled by the target process, will cause that process to terminate with a core dump. Example control messages include:
>
> Message: PCSENTRY
>
> Parameter: SYS_exit
>
> Effect: Causes monitor to be signaled when the process being monitored enters the system exit( ) routine.
>
> Message: PCSTRACE

Bennett, column 9, lines 43-60. As can be appreciated from the above excerpt, Bennett merely describes monitoring. Nowhere does Bennett say anything about the claimed action of "suspending execution of the parent process".

Applicant notes that the Examiner did not address the "suspending execution of the parent process" limitation in the Advisory Action.

(viii) Summary

In view of the foregoing, it is clear that Bennett fails to teach nearly each limitation of claim 10. Applicant therefore submits that claim 10 and its dependents are allowable over the prior art of record and requests that the Board overturn the rejections of those claims.

b. Claims 35, 38, and 39

Applicant's independent claim 35 provides as follows (emphasis added):

> 35. A computer readable memory that stores a system, the system comprising:
>
> a parent process configured to, before a vfork call is executed by the parent process, generate a pre-fork event that contains a process identifier of a child process that will be spawned from the parent process when the vfork system call is executed by the parent process; and
>
> a process monitor configured to receive the pre-fork event and process identifier before the vfork system call is executed by the parent process, suspend execution of the parent process, and generate a process monitor thread that enables observation of trace events generated by the child process.

Regarding claim 35, Bennett fails to teach a parent process configured to "generate a pre-fork event that contains a process identifier of a child process that will be spawned from the parent process" for reasons described above in subsection (iv). Again, Bennett says nothing about an identifier that indicates a child process that will be *will be* created.

In the Advisory Action, the Examiner argues that Bennett in column 8, lines 33-45 discloses generating a pre-fork event that contains a process identifier of a child process that will be spawned from the parent process. As indicated in the foregoing, that portion of Bennett's disclosure provides as follows:

> It is then left to the fault monitor 212 to determine for which processes it has already started a monitoring thread and for which processes a new

> thread should be started. In one embodiment, the fault monitor 210 sends a read call to the fault detection driver 214 as soon as it has returned and handled the prior batch of process IDs. In this way, new process IDs are returned in the next read call or immediately fulfill a blocked read call. In an alternate embodiment, receipt of a new process ID from a new process will automatically generate a new process ID message to the fault monitor 212 from the fault detection driver 214. In another alternate embodiment, the fault monitor 212 will issue a periodic read call to the fault detection driver 214 to determine if any new processes have begun.

Bennett, column 8, lines 33-45. In the above excerpt, Bennett merely describes receiving a "new process ID" from a new process, i.e., a new process that is already in existence. Furthermore, Bennett describes determining if any new processes "have begun," i.e., are already in existence. Therefore, Bennett clearly does not teach generating an event that contains an identifier of a child process that will be spawned.

With further reference to claim 35, Bennett fails to teach a process monitor configured to "receive the pre-fork event and process identifier before the vfork system call is executed", for reasons described above in relation to subsection (ii). Regarding column 6, lines 30-42, which are relied upon by the Examiner in the Advisory Action, although Bennett states that "Solaris" includes vfork( ) system calls, that statement neither teaches nor suggests the action of receiving a pre-fork event and process identifier "before the vfork system call is executed". Even if one were to assume that Bennett disclosed providing an indication when a vfork system call is executed Bennett is completely silent as to providing an indication that such a system call *is about to happen*.

As a further matter regarding claim 35, Bennett does not teach a process monitor configured to "suspend execution of the parent process". As noted above in relation to subsection (iii), Bennett's teaching of a "monitor suspension control message" that "suspends the monitoring thread" in column 10, lines 25-35 is not a teaching of suspending execution of a "parent process" as is explicitly required by Applicant's claim 35.

In further regard to claim 35, Bennett does not teach a process monitor configured to "generate a process monitor thread that enables observation of trace events generated by the child process" for reasons described above in relation to subsection (v). Regarding Bennett's column 7, lines 40-64, which were relied upon by the Examiner in the Advisory Action, Bennett fails to even mention any of a "monitor thread", a "trace event", or a "child process" in column 7, lines 40-64.

In view of the foregoing, it is clear that claim 35 and its dependents are allowable over Bennett. Applicant therefore requests that the rejections to those claims be overturned.

VIII. Conclusion

In summary, it is Applicant's position that Applicant's claims are patentable over the applied prior art references and that the rejection of these claims should be withdrawn. Appellant therefore respectfully requests that the Board of Appeals overturn the Examiner's rejection and allow Applicant's pending claims.

Respectfully submitted,

By: [illegible]
David R. Risley
Registration No. 39,345

Claims Appendix under 37 C.F.R. § 41.37(c)(1)(viii)

The following are the claims that are involved in this Appeal.

1-9. (Canceled)

10. A method for controlling the execution of a child process created from a parent process, the method comprising:

instrumenting a parent process;

monitoring execution of the parent process with a process monitor to collect information as to the run-time behavior of the parent process;

before a vfork system call is executed, receiving with the process monitor indicia from the parent process that a vfork system call will be executed by the parent process;

suspending execution of the parent process;

extracting with the process monitor a process identifier from the indicia, the process identifier identifying a child process to be generated by the parent process when the parent process executes the vfork system call;

setting with the process monitor a process monitor thread to observe trace events generated by the child process;

resuming execution of the parent process to enable the parent process to execute the vfork system call; and

again suspending execution of the parent process.

11. The method of claim 10, further comprising:

waiting for indicia that the child process has invoked at least one of an exec system call and an _exit system call or has been terminated by an operating system;

setting a process monitor thread to observe trace events generated by the parent process; and

resuming execution of the parent process.

12. (Canceled)

13. The method of claim 10, wherein receiving indicia comprises receiving a pre-fork event that includes a process identifier that identifies the child process.

14-28. (Canceled)

29. (Allowed) A method for controllably switching a target process of a process monitor thread between an instrumented parent process and a child process generated by the parent process, the method comprising:

checking whether the successful initiation of the child process can be asserted;

when the successful initiation of the child process cannot be asserted, checking if the parent process responsible for creating the child process received indicia of a failure of a vfork system call designated to create the child process by searching for a trace event while performing a non-blocking trace wait on the parent process;

when the indicia has not been received,

waiting an amount of time before rechecking for the successful initiation of the child process; otherwise,

notifying a software monitor of the unsuccessful initiation of the child process and resuming execution of the parent process;

monitoring the parent process;

otherwise, when the successful initiation of the child process can be asserted,

monitoring the successfully created child process.

30. (Allowed) The method of claim 29, wherein the step of checking whether the successful initiation of the child process can be asserted comprises verifying the success of a trace event by using the process identifier of the child process.

31. (Canceled)

32. (Allowed) The method of claim 29, further comprising:

aborting child process monitoring when the initiation of the child process is unsuccessful.

33-34. (Canceled)

35. A computer readable memory that stores a system, the system comprising:

a parent process configured to, before a vfork call is executed by the parent process, generate a pre-fork event that contains a process identifier of a child process that will be spawned from the parent process when the vfork system call is executed by the parent process; and

a process monitor configured to receive the pre-fork event and process identifier before the vfork system call is executed by the parent process, suspend execution of the parent process, and generate a process monitor thread that enables observation of trace events generated by the child process.

36. (Canceled)

37. The method of claim 10, further comprising analyzing run-time data observed during execution of the parent process once it is determined that the parent

process has generated at least one of an exec system call and an _exit system call or has been terminated by an operating system.

38. The computer readable memory of claim 35, wherein the process monitor is further configured to generate a process monitor that enables observation of the parent process upon resumption of execution of the parent process after receipt of indicia that the child process has invoked at least one of an exec system call and an _exit system call or has been terminated by an operating system.

39. The computer readable memory of claim 38, wherein the process monitor is further configured to resume execution of the parent process.

Evidence Appendix under 37 C.F.R. § 41.37(c)(1)(ix)

There is no extrinsic evidence to be considered in this Appeal. Therefore, no evidence is presented in this Appendix.

Related Proceedings Appendix under 37 C.F.R. § 41.37(c)(1)(x)

There are no related proceedings to be considered in this Appeal. Therefore, no such proceedings are identified in this Appendix.